Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computational Biology and Chemistry 28 (2004) 97–107

Computational Biology and Chemistry

www.elsevier.com/locate/compbiolchem

Identification of related gene/protein names based on an HMM of name variations

L. Yeganova*, L. Smith, W.J. Wilbur

Computational Biology Branch, National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bldg. 38A, 8600 Rockville Pike, Bethesda, MD 20894, USA

Received 4 November 2003; received in revised form 11 December 2003; accepted 12 December 2003

Abstract

Gene and protein names follow few, if any, true naming conventions and are subject to great variation in different occurrences of the same name. This gives rise to two important problems in natural language processing. First, can one locate the names of genes or proteins in free text, and second, can one determine when two names denote the same gene or protein? The first of these problems is a special case of the problem of named entity recognition, while the second is a special case of the problem of automatic term recognition (ATR). We study the second problem, that of gene or protein name variation. Here we describe a system which, given a query gene or protein name, identifies related gene or protein names in a large list. The system is based on a dynamic programming algorithm for sequence alignment in which the mutation matrix is allowed to vary under the control of a fully trainable hidden Markov model.

© 2003 Published by Elsevier Ltd.

Keywords: Automatic term recognition; Gene name variation; Hidden Markov model; Information extraction

1. Introduction

1.1. Background

Identification and classification of named entities in scientific text is a prerequisite for automatic extraction of knowledge from literature (de Bruijn and Martin, 2002). Although named entity recognition might be regarded as a solved problem in some domains, it still poses a significant challenge in the area of molecular biology. Several researchers have looked at the problem of identifying gene or protein names in molecular biology texts (Collier et al., 2000; Franzen et al., 2002; Fukuda et al., 1998; Hanisch et al., 2003; Krauthammer et al., 2000; Narayanaswamy et al., 2003; Proux et al., 1998; Tanabe and Wilbur, 2002). A large number of different methods have been applied to this problem, including part-of-speech tagging, hidden Markov models, decision trees, Bayesian methods, rule based systems, regular expressions, and a variety of knowledge based resources. The problem of determining when two differing strings represent the same gene or protein seems to have received much less attention. It is to this problem of name variation, not unrelated to the first problem, that we wish to give attention here. In fact, we believe it is useful to relax the criterion in the name variation problem somewhat and ask how to determine when two differing strings represent related genes or proteins. If we can solve this relaxed problem then we may be able to apply that solution to make progress on the named entity recognition problem. Genes and proteins tend to come in families or related groups, and if we can compile a reasonable list of genes and recognize their variants then this capability may be used to identify the numerous variations of gene or protein names seen in free text.

Automatic term recognition (ATR) has received attention for general text (Frantzi et al., 2000; Hahn et al., 2001; Jacquemin, 1994; Jacquemin, 2001; Nenadic et al., 2002; Nenadic et al., 2003). Jacquemin (2001) has described the problem as that of representing and processing morphological, syntactic, and semantic variations. Methods that have been applied in the general case have also been applied to the problem of gene name variation detection. One type of gene name semantic variation involves gene name abbreviations and their corresponding full forms. Methods have been developed to relate gene name full forms to their abbrevi-

* Corresponding author. Tel.: +1-301-402-0776; fax: +1-301-480-2290.
E-mail address: yeganova@ncbi.nlm.nih.gov (L. Yeganova).

1476-9271/$ – see front matter © 2003 Published by Elsevier Ltd.
doi:10.1016/j.compbiolchem.2003.12.003

ations (Liu and Friedman, 2003; Yu et al., 2002). Another approach allows the identification of gene name synonyms based on the recognition of certain lexico-syntactical patterns in which they co-occur (Yu and Agichtein, 2003; Yu et al., 2002). Both of these methods use the context of occurrences in actual text and find related gene names that would be difficult or even impossible to recognize based on name string comparisons alone. These are important methods for recognizing semantic name variations (unrelated string, same meaning). However, our interest is in how to recognize those name variations that are reflected in the underlying strings being compared.

Name variations that are reflected in the underlying strings are generally morphological (related by derivation or inflection), as seen in the relatedness of *recognition* and *recognize*, syntactical (different grammatical constructions with the same meaning and related underlying words), as seen in the relatedness of *lung cancer* and *cancer of the lung*, or some one of a host of poorly characterized variations that occur sporadically. This later category would include such variations as *p53* versus *p53 gene*, *caspase-3* versus *casp3*, or *bcl2* versus *bcl-2* versus *bcl 2*. Terms may differ in token separators and punctuation, have additional or missing characters, be misspelled, truncated, or abbreviated. As another example, *abl1-bcr gene*, *abl1/bcr gene*, *bcr-abl1 gene*, and *5'abl1* all refer to the *abl* gene. Few methods for handling such term variations have been developed. A Basic Local Alignment Search Tool (BLAST)-based system presented by (Krauthammer et al., 2000) uses approximate string matching techniques and dictionaries to recognize spelling variations in gene or protein names. They have encoded gene names and text in terms of the nucleotide alphabet and have used BLAST to look for 'homologies' between a query gene name and the text. A similar problem was addressed by (Cohen et al., 2002) who studied contrast and variability in gene names to develop heuristics to distinguish between gene or protein names with different meaning from names that are synonyms. They found that capitalization could be ignored, parenthesized material and hyphens were optional, and vowels tended to be interchangeable. However, they did not implement a system based on these observations.

There are other works that are related less to the problem that we pose, nevertheless the ideas employed may be useful for our problem. For example, (Hahn et al., 2001) approached the problem of morphological variations for medical document retrieval in German by segmenting query and database terms into medically plausible subword units that are morphemes or their combinations. One of our approaches will adapt this idea of subwords and apply it to gene name recognition.

1.2. Overview

We studied a corpus of gene names to find potential methods for identification of related gene or protein names based on their underlying strings. Among several methods tested we focused our attention on the following: character *n*-gram methods and hidden Markov models (HMM). Below we give a motivation for using these methods and an overview of the related literature.

Attempts have been made in the past to use character *n*-grams derived from the words of a document to represent that document for the purpose of similarity matching and retrieval. Particularly, the method was utilized by (Damashek, 1995) in an attempt to perform a text categorization task. However, this approach has been judged as not competitive with other more standard approaches (Salton, 1995; TREC-program-committee, 1995). The character *n*-gram method was later used by (de Bruijn et al., 2000) who compared it with word-based information retrieval methods and again found that word based methods were consistently better than character *n*-grams. The problem encountered in both cases was that *n*-grams are too nonspecific for the words they represent, and given a document with a large number of words, enough *n*-grams will be generated in common with an unrelated document to cause ambiguous retrieval results. For example the character trigram *fix* is contained in *suffix*, *prefix*, *infix*, and *fixture*, etc. However, (Kim and Wilbur, 2001) demonstrated that more useful results can be obtained by applying this method to shorter documents, i.e. phrases. The reason for this is that a few words in a phrase are not sufficient to allow much ambiguous matching. Following them, we have implemented *n*-gram phrase matching algorithms. This proves beneficial because gene names are represented by reasonably short phrases.

As mentioned above, one could view the problem of identifying related gene or protein names as a biological sequence alignment as in (Krauthammer et al., 2000), and moreover, one could treat names in their original alphabet. The standard approach would apply dynamic programming algorithms (Gotoh, 1982; Needleman and Wunsch, 1970) that align two sequences of symbols, achieving a minimal cost. The cost of an alignment is the sum of the costs of the symbols paired by the alignment, where one of the symbols may be a gap. These costs depend on a predetermined mutation matrix of symbol-pair cost values, and hence the usefulness of the resulting alignment depends on how accurately the model reflects the nature and origin of the sequences involved. Later, the problem has been reformulated in terms of a random process with a probability model by (Durbin et al., 1998). He proposed using Hidden Markov Models for pairwise sequence alignments: just as a standard HMM can generate a single sequence of symbols, a pair HMM can generate an aligned pair of sequences. He proposed a three-state HMM, where one state corresponds to matches, and the other two states correspond to insertions to the first and second sequence, respectively. Each subsequent state is chosen according to the distribution of transition probabilities leaving the current state, and a symbol-pair to be added to the alignment is chosen according to the emission distribution in the new state. The challenge of the approach is to get reasonable estimates for transition and emission probabilities, for a given state network. Smith et al. (2003) provided an advance by presenting a training procedure for the pair HMM which is unique in that one specifies a collection of pairs of sequences without any corresponding alignments. One assigns initial parameter values for transition and emission probabilities (for example, random or uniform), which are iteratively updated during the training to produce maximum likelihood estimates based on the set of training pairs. The resultant probabilities govern state transitions and output of paired or gapped sequence elements.

In this work we develop a training set of related gene name pairs and use it to train several pair HMM models. The best of these models has proved to be the most sensitive method we have found to compare gene names and detect related pairs. However, the run time of the HMM models is long and we found a substantial time savings by combining the best of the HMM models with the best of the *n*-gram methods briefly described above. Section 2 of the paper describes the data and preparation of the data sets. Section 3 provides more detail on the recognition methods that we use, including the pair HMM, BLAST, and *n*-gram phrase matching algorithms. Section 4 gives evaluation results, and we conclude with a discussion in Section 5.

2. Data source and preparation

2.1. Creating data sets

We have obtained a list of 32,000 gene and protein names from LocusLink (Pruitt and Maglott, 2001) and the Gene Ontology Consortium (Ashburner and Lewis, 2002; Consortium, 2000). In the scope of this research we did not differentiate between gene and protein names, hence they all are referred as gene names. Out of that list we chose the names occurring in at least 100 MEDLINE documents in order to have enough statistical data for Bayesian learning, but in no more then 10,000 MEDLINE documents, as very high frequency names may be uninformative. From the resulting set of 3,754 gene names we chose 90 (listed in Appendix A), which we will refer to as query gene names or $\{qn_i\}_{i=1}^{90}$. These 90 names were chosen randomly, except the numbers have been chosen to represent names starting with every letter of the alphabet proportional to their distribution in the original set. For instance, the original set contained 124 gene names starting with letter ‘B’, and proportionally our sample contains three names starting with ‘B’. Gene names starting with letter ‘A’ are overrepresented due to the fact that they were already processed for a preliminary study (Smith et al., 2003), and we have included all of them.

For each query gene name, the goal was to identify closely related terms from a list of potential gene names. First, we retrieved MEDLINE abstracts related to the gene name by contents, but not containing that name spelled exactly as it appears in the query. To do this, we used Naïve Bayesian learning (Mitchell, 1997) to learn the difference between the abstracts that contain the query gene name and the remaining MEDLINE abstracts that do not. The later abstracts were then scored according to the log odds of being related to the abstracts containing the query gene name and placed in rank order. High scoring documents are more likely to discuss the subject of the query gene name but under a different name. We limited our consideration to the 10,000 top scoring abstracts.

The next step was to apply the ABGene tagger (Tanabe and Wilbur, 2002, and freely available at ftp.ncbi.nlm.nih.gov/pub/tanabe) to the above chosen abstracts to extract potential gene or protein names, thus creating for each query gene name qn_i a corresponding set $GN_{qn(i)}$ that contained all these potential gene names. This extracted set of potential names generally contained many terms unrelated to the query gene name. In order to locate the terms in $GN_{qn(i)}$ that are closely related to the query name qn_i we first applied a crude algorithm. A phrase in $GN_{qn(i)}$ was considered to be potentially related to qn_i if it satisfied one of the conditions:

- the phrase contained all the tokens of the query gene name in any order;
- the phrase contained an abbreviation of the query gene name from a standard list of abbreviations (McCray et al., 1994).

Non-alphanumeric characters were ignored at this step. This process selected most of the data of interest. Finally, we manually reviewed all the data to correct possible false positives and false negatives. The result was a subset $RN_{qn(i)}$ of $GN_{qn(i)}$ for every qn_i that contained phrases that were humanly judged related to the query gene name qn_i. The criterion of relatedness we used was that if an interest in qn_i by an investigator would likely imply an interest in $x \in GN_{qn(i)}$ then x was considered a member of $RN_{qn(i)}$. The set of all relevant phrases in $\{RN_{qn(i)}\}_{i=1}^{90}$ comprised $<$1% of the phrases in $\{GN_{qn(i)}\}_{i=1}^{90}$.

In summary we have obtained a set of query gene names to be examined, $QN = \{qn_i\}_{i=1}^{90}$, and corresponding phrase sets $GN_{qn(i)}$ and $RN_{qn(i)} \subset GN_{qn(i)}$, $i = 1, \ldots, 90$ where $RN_{qn(i)}$ denotes the set of phrases among $GN_{qn(i)}$ that were judged to be related to the query name qn_i. As an example, one of the queries, *coproporphyrinogen oxidase*, has 13 related phrases, which are:

coprogen oxidase, coprogen oxidase gene, coprogen oxidase rna, coproporphyrinogen iii oxidase, coproporphyrinogen iii oxidases, enzyme coproporphyrinogen iii oxidase, lymphocyte coproporphyrinogen iii oxidase, oxygen-independent coproporphyrinogen iii oxidase, oxygen-independent coproporphyrinogen iii oxidases, coproporphyrin iii oxidase, coproporphyrin oxidase, oxygen-independent coproporphyrinogen iii, oxygen-independent coproporphyrinogen iii dehydrogenases.

Another query *11beta hsd* has 495 related phrases, including:

11 beta hsd, 11 beta hsd2 protein, 11 beta hydroxysteroid dehydrogenase, 11 beta-hydroxysteroid dehydrogenase-1, 11 beta-hydroxysteroid dehydrogenase type 1, 11 betahsd2 enzyme, v11-beta-hydroxysteroid dehydrogenase, 11beta-hsd2 gene, 11betahsd1 dehydrogenase, enzyme 11 beta-hsd, inhibiting 11beta-hydroxysteroid dehydrogenase type 2, type 1 11 beta-hsd, type 2 11 beta-hsd mrna, 1lbetahsd isozymes.

2.2. Training/testing sets

To estimate the accuracy of the methods used, we have performed three-fold cross-validation based on the three training and testing sets that we have created as described below. The set of 90 indices was randomly partitioned into three sets of thirty indices each, I_1, I_2, and I_3. The first training set was based on the 60 $RN_{qn(i)}$, such that $i \in I_1 \cup I_2$. For each i, let $n_i = ||RN_{qn(i)}||$ (here $||X||$ denotes the number of elements in *X*). Every phrase in each $RN_{qn(i)}$ was paired with the remaining phrases in that same $RN_{qn(i)}$, resulting in $N_i = n_i(n_i - 1)/2$ different pairs for the set $RN_{qn(i)}$. To avoid over representing gene names with many variants in the training sets, all related pairs were taken when $N_i \leq 300$, but otherwise only 300 were randomly sampled from the set of N_i. For example, gene name *coproporphyrinogen oxidase* has 13 related phrases which generate 78 related pairs, among them:

coproporphyrin oxidase;
oxygen-independent coproporphyrinogen iii.

coprogen oxidase rna;
coproporphyrinogen iii oxidase.

enzyme coproporphyrinogen iii oxidase;
coproporphyrin iii oxidase.

On the other hand, gene name *11beta hsd* has 495 related phrases generating 122,265 pairs from which 300 were randomly sampled.

The test set for the first training set was created using all of the $GN_{qn(i)}$ corresponding to the remaining 30 query names, i.e. $GN_{qn(i)}$, $i \in I_3$. A given query name qn_i, $i \in I_3$, was paired with all the phrases $x \in GN_{qn(i)}$, and the pairs were marked as positive examples if $x \in RN_{qn(i)}$ and negative examples otherwise. Our methods were rated on how well they succeed in correctly classifying the members of the test sets.

The remaining two training and test sets were created in like manner. The resulting training data sets on the average consist of approximately 13,000 pairs of related phrases. Each test set contained about 823,500 possibly related gene or protein name pairs distributed at an average rate of about 27,450 pairs per query name over the 30 query names in the test set. The training sets were used to train the HMM models. The other methods did not require training. All methods were tested on the three test sets, and the results were averaged.

3. Recognition methods

Here we provide details of the methods we used to identify related gene names. The methods are Hidden Markov Models (HMM), BLAST, and *n*-gram Phrase Matching Algorithms of several types. These algorithms can be used separately, or may, in some cases, be combined to decrease run time or improve accuracy.

3.1. Hidden Markov Model

The parameters needed to define an HMM are its states, transition probabilities, and emission probabilities. We first describe the elements of an HMM designed to output a linear sequence. Let *N* denote the number of states of a hidden Markov model, with initial state probabilities $\{\pi_i\}$, and transition probabilities $\{a_{ij}\}$. Let the alphabet for output be $S = \{u_k\}_{k=1}^{M}$ and the probability of u_k as output from state *i* be denoted by b_{ik}. Let $X = \{u_{k(j)}\}_{j=1}^{L}$ denote an observed sequence. Then the *Viterbi algorithm* allows one to compute the probability of the most likely path producing *X* as output. The algorithm begins in position 0 without any output and with a set of probabilities $p_{0,i} = \pi_i$ and recursively defines $\{p_{t,i}\}$ by

$$p_{t+1,i} = \max_j p_{t,j} a_{ji} b_{ik(t+1)} \tag{1}$$

The maximal probability path is defined by that *j* for which $p_{L,j}$ is a maximum and the state sequence or path is recoverable by setting backward pointers at each step in (1). The *forward algorithm* computes the sum of the probabilities of producing *X* as output over all possible paths that the process could take. It is identical to the Viterbi algorithm except for the recursion (1), which becomes

$$p_{t+1,i} = \sum_j p_{t,j} a_{ji} b_{ik(t+1)} \tag{2}$$

In this case the overall probability of observing the output sequence *X* is $\sum_j p_{L,j}$.

Thus far we have dealt with a hidden Markov model designed to output a simple linear sequence. The situation for hidden Markov models that output a sequence alignment is slightly more complicated. In this case the alphabet *A* from which sequence elements must be chosen combines with an element *g* representing a gap to form the alphabet for output according to

$$S = (A \cup \{g\}) \times (A \cup \{g\}) - \{(g, g)\} \tag{3}$$

Now instead of moving linearly along a sequence one is moving over a grid in two dimensions. Instead of looking back from any point $t + 1$ in a sequence to t as in (1) and (2), one now looks back from any point $(r + 1, s + 1)$ on the grid to three possible points $(r+1, s)$, $(r, s+1)$ or (r, s) from which one may have come. Optimization in (1) or summation in (2) is now not only over the states but over the three prior points from which one may have come. There are slight modifications at the origin and along the sides of the grid in that one has fewer directions to look back. Whereas for a sequence the traversal is completed when one reaches the end of the sequence, on the grid the traversal is completed when one reaches the corner opposite from the origin where one began. Movements horizontal or vertical correspond to output of a gap in one sequence and the next element of the other sequence, whereas movement on the diagonal corresponds to output of the next element of each sequence as a pair. For more details regarding general HMMs see (Rabiner, 1989; Charniak, 1993), while further details regarding pair HMMs may be found in (Durbin et al., 1998; Smith et al., 2003).

To train the HMM we have used the extension of the Baum–Welch algorithm applicable to pairs of related sequences, as described in (Smith et al., 2003). Given a training set of pairs of sequences that are considered related, the training algorithm produces maximum likelihood estimates for the HMM transition and emission probabilities. The results are strongly dependent on the state structure assumed for the HMM. While the Baum–Welch algorithm (a form of EM algorithm) can only guarantee a locally optimal solution, the results are generally found useful.

Once the HMM has been trained, the forward algorithm can be used to compute the probability of observing a given pair of strings (in our case phrases), and the Viterbi algorithm can be used to find the most likely path though the states and a corresponding alignment. Either one, the probability of observation of a pair of phrases or the probability of the most likely path can be used to construct a score measuring the relatedness of two phrases. To test the scoring produced by the forward algorithm, for each pair (qn_i, x), $x \in GN_{qn(i)}$ the probabilities of the pairs (qn_i, x) and $(null, x)$ are calculated and the score is defined as the difference

$$score(x) = \log_{10} \Pr(qn_i, x) - \log_{10} \Pr(null, x) \tag{4}$$

where *null* is the empty string. Scoring produced by the Viterbi algorithm is the same except the probabilities of the observation of pairs are replaced by the probabilities of the optimal paths corresponding to those pairs.

It is important to recognize what the pair HMM models do not learn. They do not learn to recognize the query sequences $\{qn_i\}_{i=1}^{90}$, for example. Thus, it is not so important what the query gene names are, but it is very important that they be common enough to exhibit many variations in form. Intuitively, the objective of this method is to learn from a large sample of name pairs what is not important in a gene name, as opposed to what signals a difference in meaning. The pair HMM models learn to assign a high probability to those pairs designated related and to any pairs that are related in a similar manner. This knowledge is captured in the state transition probabilities and in the character pair output probabilities. Some characters are less critical to the meaning of gene name phrases, particularly punctuation. Note that in this setting we treat gene names as sequences in their original alphabet, which, in addition to letters, digits and the space, includes the following special characters: []{}(),.”’*&/< >−+.

3.2. BLAST

BLAST, is a fast partial matching algorithm used for DNA and protein sequence comparison introduced by (Altschul et al., 1990). It compares a query sequence to all the sequences in a specified database and assigns a score to each sequence that reflects the degree of similarity with the query. Krauthammer et al. (2000) noted that text string matching and DNA sequence comparison are related problems with the exception of the alphabets used. The problem that they addressed was how to identify gene and protein names in journal articles. They proposed encoding text characters using the nucleotide alphabet, by substituting each character with a predetermined four-letter nucleotide combination. When both the query name and the text to be searched are thus encoded, BLAST may be used to query the gene name against the database of scientific articles.

While our problem of finding related gene names is somewhat different, clearly the same idea can be applied to it. To use BLAST we have translated each query name $qn_{(i)}$ and all of the phrases in $GN_{qn(i)}$ into the nucleotide alphabet (‘A’, ‘C’, ‘G’, and ‘T’) representation according to the conversion table given in (Krauthammer et al., 2000). Now each letter is represented by a four-letter combination. For example, letter ‘A’ corresponds to ‘AAAC’, letter ‘B’ corresponds to ‘AAAG’, the space character corresponds to ‘ATCC’, etc. Thus the gene name *tnf beta* is represented as ACGG|ACAG|AACT|ATCC|AAAG|AACG|ACGG|AAAC (bars added for clarity).

BLAST outputs phrases that potentially match the query phrase along with a raw score *S* and an expectation value (*E*-value). The *E*-value is defined as the number of different alignments with raw scores greater than or equal to *S* that are expected to occur in the database search by chance. The lower the *E*-value, the more significant the match. We take the negative of the *E*-value as the score for our purposes. BLAST only outputs the phrases that have an *E*-value less than some specified threshold. We have chosen this threshold to be 1,000, which allows us to obtain a sufficient number of phrases in the output. BLAST also requires a setting for word size that represents the minimal size of pieces that must agree in order to detect a match within a pair of phrases. We have chosen a word size of eight in the nucleotide alphabet for increased sensitivity.

3.3. n-Gram phrase matching

Another approach to identification of phrases related to a query gene name is based on flexible phrase based query handling algorithms (Kim and Wilbur, 2001). These algorithms decompose phrases into smaller pieces, substrings or *n*-grams, and create a so-called document or bag-of-substrings (in analogy with bag-of-words) representation for each phrase. The relatedness of two phrases is scored based on the *n*-grams they have in common and generally employs a weighting scheme for the *n*-grams. Phrases may be decomposed under different rules with quite different results and we investigate three different approaches.

3.3.1. n-GRM

In this method (Wilbur and Kim, 2001) phrases are lowercased and broken into words at spaces. Assuming that *n* is three, individual words are broken into trigrams in such a way that strings of length *L* produce (*L-2*) overlapping triplets of letters. If the word length is less than three then the whole word is taken as the only trigram produced. In this method the beginning of a word is given more importance than the remainder, as the first trigram appears in the document in two different forms, twice marked with an added '!' character at the end and once without. Also, the first character of each word appears alone, and for any two consecutive words the first letters are used in occurrence order with a space character between to form a so-called bridging trigram. For example the document produced by 3-GRM decomposition for the gene name *tnf beta* is: tnf, bet, eta, tnf!, tnf!, bet!, bet!, t#, b#, t b.

3.3.2. n!-GRM

In this method, phrases are lowercased, all the characters except letters and digits are removed, and the remaining characters are consolidated into a single string which is treated as a whole. The resultant string is decomposed into overlapping *n*-grams, $(n-1)$-grams, ..., 2-grams formed from consecutive characters in such a way that strings of length *L* produce $(L-1)$ bigrams, $(L-2)$ trigrams, ..., $(L-(n-1))$ *n*-grams. In this case, each piece is added only once to the document being produced though duplicates may arise from different parts of the string. For example, the document produced by *3!*-GRM decomposition for the gene name *tnf beta* is: tnf, nfb, fbe, bet, eta, tn, nf, fb, be, et, ta.

3.3.3. SUB-word

A different approach to segmentation of phrases can be based on what we might call the natural occurrence within the data of segments as words. For any phrase *ph* let $A(ph)$ denote the set of all contiguous alphanumeric substrings of *ph* that have ends that either coincide with the ends of *ph* or are demarcated by non-alphanumeric characters in *ph*. Given a set of phrases Π let Λ denote the set of alphanumeric phrases defined by

$$\Lambda = \bigcup_{ph \in \Pi} A(ph) \tag{5}$$

The set Λ will function for us like a lexicon. Again let *ph* denote a phrase and suppose that $A(ph) = \{an_i\}_{i=1}^{k}$ where the segments an_i are assumed to be numbered in the order that they occur in *ph*. Then define $B(ph)$ as the set of all elements *bn* of Λ where for some $1 \leq p \leq q \leq k$, $bn = an_p, \ldots, an_q$ where we understand concatenation on the right side of this equality when $p < q$. Generally $B(ph)$ will be a richer representation of *ph* than $A(ph)$. Again let $C(ph)$ represent all those elements of Λ which occur as contiguous substrings of elements of $B(ph)$. Then $C(ph)$ will generally have more elements than $B(ph)$. We will let SUB1 stand for the approach in which we represent each *ph* in Π by the elements of $B(ph)$ with a count equal to the number of ways it is generated from the elements of $A(ph)$. Correspondingly SUB2 will represent the approach that is the same as SUB1 except when *ph* is the query phrase it will be represented by the elements of $C(ph)$ each with a count equal to the number of ways it is generated from the elements of $B(ph)$. Thus in SUB2 the phrases in the database all have the same representation as in SUB1, but the query phrase is given a potentially richer representation. However, all strings used in representations are found in Λ.

3.3.4. Vector scoring

The different methods of representing a phrase as a document are used to compare phrases using vector retrieval formulas. Let *t* denote a feature (substring) which appears in at least one document representing a phrase in $\Pi = QN \cup \bigcup_{i=1}^{90} GN_{qn(i)}$. Then there is associated with *t* a global weight gw_t given by the formula

$$gw_t = \log\left(\frac{N}{n_t}\right) \tag{6}$$

where *N* is the number of phrases in the database Π and n_t is the number of the documents representing members of Π that contain *t*. Likewise if *p* is any phrase *t* has a local weight in *p* given by

$$lw_{pt} = \log(ft_p + 1) \tag{7}$$

where ft_p is the number of times *t* occurs in the bag-of-substrings representation for *p*. Such weights allow us to represent any phrase *p* as a vector

$$v_p = (v_{pt})_{t \in \Lambda} \tag{8}$$

where

$$v_{pt} = lw_{pt}\sqrt{gw_t}. \tag{9}$$

For any phrases *p* and *q* we may then compute the similarity between *p* and *q* by

$$\mathrm{sim}(p, q) = \frac{v_p \cdot v_q}{\sqrt{v_p \cdot v_p} \cdot \sqrt{v_q \cdot v_q}} \tag{10}$$

which is the standard cosine similarity formula as given in (Salton, 1989). This formula is applied to all the different methods of representing phrases by substrings presented here as *n*-gram methods. Given a query qn_i in one of the test sets, we compute $\mathrm{sim}(qn_i, x)$ for all $x \in GN_{qn(i)}$ and treat the $\mathrm{sim}(qn_i, x)$ as scores in an attempt to differentiate the *x* related to qn_i from the *x* unrelated to qn_i.

3.4. Combining methods

There is often a tradeoff between speed and accuracy. In an attempt to obtain the advantage of both, we have used the combination of fast and slow methods in the following way: a computationally 'cheaper' method is used to score all the pairs of phrases, and only after that another, more accurate, method is applied to the T top scoring phrases extracted by the first method. As we will show, this approach proves to be very useful, and hence we will consider the combination of methods.

3.5. Evaluating

To evaluate and compare all the different methods, we have used standard recall and precision figures. Recall is computed as the ratio of good phrases retrieved to the total number of good phrases in the system. Precision is defined as the ratio of the number of good phrases retrieved to the total number of phrases retrieved.

$$\text{Recall} = \frac{\text{number of relevant phrases retrieved}}{\text{total number of relevant phrases}}$$

$$\text{Precision} = \frac{\text{number of relevant phrases retrieved}}{\text{number of phrases retrieved}}$$

Recall and precision are the simplest and most frequently used measures of performance (Salton, 1992). For each method we have estimated recall and precision values at three different depths of retrieval, the top 20, 100, and 1,000 ranks. For recall and precision individual values are computed for each query term, and then the individual values are averaged over all query terms to produce so-called macro-averaged values (Manning and Schutze, 1999), and it is these macro-averages we report.

4. Evaluation

4.1. HMM results

We have trained three Hidden Markov models with different state structures. The first is a one state model, it only trains emission probabilities of a single state and there are no state transitions. The other two models, 'pure' (HMM-3P) and 'mixed' (HMM-3M), are three state models, where state 2 outputs only gaps in the first string, and state 3 outputs only gaps in the second string. In the HMM-3P model, state 1 outputs only matches, whereas in the HMM-3M model, state 1 outputs both matches and gaps. In the three-state models, both transition and emission probabilities are trained. All of these models are symmetric and do not allow mismatches.

Given the trained model, the forward algorithm is used to compute the probability of observing a given pair of phrases. This probability is then used to construct a score measuring the relatedness of two phrases as given in Eq. (4). This allows us to carry out a retrieval experiment testing how well we could identify the names judged to be related to the query gene or protein names. For each HMM model, we have performed three-fold cross validation and calculated macro-averaged recall and precision values that are given in Table 1.

Table 1
Average recall and precision for three-state 'mixed', three-state 'pure', and one-state models at cut-off ranks of 20, 100, and 1,000

20	HMM-3M	HMM-3P	HMM-1
Recall	0.448	0.418	0.198
Precision	0.706	0.684	0.324
100	HMM-3M	HMM-3P	HMM-1
Recall	0.691	0.649	0.404
Precision	0.412	0.376	0.217
1000	HMM-3M	HMM-3P	HMM-1
Recall	0.909	0.898	0.773
Precision	0.100	0.096	0.070

Score calculation based on the forward algorithm.

Three state models far outperform the one state model, and the mixed three state model performs significantly better than the pure three state model. The main concern with the three state models is their long run time, which is a consequence of the number of states and the length of the phrases. The HMM-3M model with the forward algorithm took about 10 h of processing time on a 500 mHz Intel Pentium 3 with 4 Gigabytes of RAM on the full set of 90 queries. To improve the speed, we have tried several techniques. One approach was to use the Viterbi algorithm instead of the forward algorithm to score the pairs of phrases. The score is calculated in a similar manner, except log-probabilities of observations are replaced with the log-probabilities of the optimal paths detected by the Viterbi algorithm. Table 2 presents these results.

In terms of run time, the Viterbi algorithm is roughly three times as fast as the forward algorithm. The Viterbi algorithm took on average 2.2 min per query, where each query phrase was compared on average against 27,450 phrases. This is getting into the practical range. Interestingly, the average accuracy of Viterbi-based scoring outperforms the accuracy of forward-based scoring in most cases. Especially for the

Table 2
Average recall and precision for three-state 'mixed', three-state 'pure', and one-state models at cut-off ranks of 20, 100, and 1,000

20	HMM-3M	HMM-3P	HMM-1
Recall	0.450	0.429	0.269
Precision	0.708	0.692	0.433
100	HMM-3M	HMM-3P	HMM-1
Recall	0.691	0.671	0.508
Precision	0.410	0.383	0.285
1000	HMM-3M	HMM-3P	HMM-1
Recall	0.914	0.897	0.821
Precision	0.103	0.098	0.078

Score calculation based on the Viterbi algorithm.

Table 3
Average recall and precision of *n*-gram phrase matching approaches and of BLAST at retrieval cut off ranks of 20, 100, and 1,000

20	4!-GRM	3!-GRM	4GRM	3GRM	BLAST	SUB1	SUB2
Recall	0.397	0.394	0.262	0.322	0.345	0.287	0.183
Precision	0.582	0.583	0.440	0.519	0.540	0.542	0.344
100	4!-GRM	3!-GRM	4GRM	3GRM	BLAST	SUB1	SUB2
Recall	0.642	0.646	0.423	0.569	0.521	0.482	0.369
Precision	0.360	0.364	0.259	0.338	0.282	0.315	0.259
1000	4!-GRM	3!-GRM	4GRM	3GRM	BLAST	SUB1	SUB2
Recall	0.915	0.917	0.715	0.869	0.837	0.673	0.684
Precision	0.104	0.104	0.081	0.092	0.088	0.065	0.075

1-state model, Viterbi-based scores produce a considerably better ranking than forward-based scores. Clearly using as a score the probability of the optimal path through the pair of phrases is advantageous in terms of both run time and accuracy.

4.2. *n*-Gram, subword segmentation, and BLAST results

All the query gene names and test set phrases have been decomposed with *n*-gram phrase matching algorithms and subword segmentation methods to obtain document representations. These are then used to score the pairs of phrases for similarity according to Eq. (10). The query gene names and test set phrases have also been encoded into the nucleotide alphabet, and the BLAST algorithm applied to score phrase pairs for relatedness. The results, parallel to those given for the HMM models, are given in Table 3.

4.3. Combining methods

As we have noted above, the HMM models, especially HMM-3M, perform very well except for a long run time. Using the Viterbi algorithm instead of the forward algorithm considerably improves the run time, but HMM-3M still requires an amount of time that would make an online implementation impractical. We have therefore combined HMM-3M with a cheaper, i.e. much faster, method. For this purpose, we have found *3!*-GRM matching to demonstrate the best results, followed by *4!*-GRM matching. We first perform *3!*-GRM and then apply HMM-3M to the top 1000 results coming from *3!*-GRM retrieval. This on the average reduces the amount of work that HMM-3M has to perform in this particular experiment by a factor of more than 27 times (a query phrase is, on the average, compared with 27K phrases). Table 4 presents the results of *3!*-GRM/HMM-3M and not only is run time decreased but accuracy is also improved in all cases over HMM-3M alone. The other methods listed in Table 3 were all tried in place of *3!*-GRM, but none of them performed as well.

To further assess the performance of the combined algorithm *3!*-GRM/HMM-3M we applied it to find names related to a query name among all the gene/protein names that appear as features of sequences in GenBank (Benson et al., 2003). We constructed a database of 1.41 million such names from GenBank. Query strings were chosen from a set of 1.14 million putative gene names extracted from MEDLINE (Tanabe and Wilbur, 2004). A set of 120 strings were randomly chosen and 107 of these were found to represent valid gene names. These were all searched against the GenBank list and good results were found for 80% of them. The remaining 20% were examined to ascertain the reason for failure. The problem in most cases was a query string consisting of a short specific term combined with a long nonspecific term such as *alleles*, *gene*, *binding site*, *factor*, *protein*, etc. For example *lxd alleles* does not occur in the database and when used as a query retrieves many strings containing the term *alleles*. The problem is cured by searching with *lxd* as this retrieves a relevant record at the third rank. In most cases where a direct query fails and general terms can be removed without affecting the identity of the gene represented it is helpful to remove them. There are other problems that can occur. For example, searching with the term *yeast pdr5* is not as effective as searching with the inverted *pdr5 yeast*. Also there are strings like *abl* that occur in words like *probable*, *thermostable*, or *variable* that are common components of gene names in GenBank. As a result a query with *abl* does not retrieve relevant material. In such cases one may find relevant material by first finding all those database entries that contain the string *abl* and then ordering them by the reciprocal of their length. This is a strategy that can replace *3!*-GRM when the latter fails.

Table 4
Results of combining *3!*-GRM matching and HMM-3M methods

20	HMM-3M	3!-GRM	3!-GRM/HMM-3M
Recall	0.446	0.394	0.449
Precision	0.701	0.583	0.710
100	HMM-3M	3!-GRM	3!-GRM/HMM-3M
Recall	0.687	0.646	0.689
Precision	0.404	0.364	0.412
1000	HMM-3M	3!-GRM	3!-GRM/HMM-3M
Recall	0.911	0.917	0.917
Precision	0.102	0.104	0.104

A ranking is first produced by *3!*-GRM and the top 1000 ranks are then re-scored by HMM-3M to produced the results reported. For ease of comparison, the results of HMM-3M and *n!*-GRM methods from Tables 3 and 4 are also included.

5. Discussion

One of the interesting and unexpected results found in this study was that, though the training of the pair HMM is done to maximize the forward probabilities of the training set pairs, the Viterbi algorithm actually gave the best results in scoring the test sets. We believe this may be explained by the fact that the forward algorithm actually sums up the probability over all possible alignment paths for two sequences and most of these alignments will make no sense semantically, whereas the Viterbi algorithm computes the probability of the most probable path and this will in many cases be the path that is most semantically meaningful. Thus perhaps it is not so surprising that the Viterbi algorithm gives the better results. Fortunately in our setting Viterbi is a faster algorithm because normalization can be used and underflow avoided without resorting to a logarithmic representation of numbers.

Another point of interest is that the two methods, HMM-3M and *3!*-GRM, complement each other. The HMM-3M aligns two phrases by matching segments of the phrases appearing in both strings. In this process no account is taken of whether these segments are meaningful or not. On the other hand, *3!*-GRM is able to measure the importance of different trigrams by their frequency throughout the whole set of strings under consideration. Since the more rare segments are more semantically significant, *3!*-GRM is able to eliminate from consideration some pairs of strings that would otherwise confuse the HMM-3M algorithm. Yet *3!*-GRM is not competitive with HMM-3M in producing the best ordering at the top ranks. In particular even though *3!*-GRM was able to get about 91.7% of good documents into the top 1,000 (recall at 1,000), it did not do a very good job at arranging them: only a 58.3% precision is achieved at the cut off rank of 20, which is on the average equivalent to 11.6 relevant phrases in the top 20. But when HMM-3M was applied to the top scoring 1,000 documents from *3!*-GRM, precision at 20 was improved to 71%, which is on the average equivalent to 14.2 relevant phrases in the top 20. Evidently both the recall and precision achieved by combining the two methods are better than those for HMM-3M alone because *3!*-GRM is able to rule out of the top 1,000 ranks some of the phrases that HMM-3M would have ranked high. One is led to speculate that an ideal algorithm for the task we study would combine in a seamless way the virtues of *3!*-GRM and HMM-3M. The form such an algorithm might take is a subject for future research.

The combination *3!*-GRM/HMM-3M works quite well for many of the morphological, syntactical, and irregular string variations that are encountered in the nomenclature of genes and proteins. However, there are significant ways in which it could be improved and its scope broadened. First, the algorithm has no ability to recognize that *lymphocyte associated receptor of death*, *LARD*, *Apo3*, *DR3*, *TRAMP*, *wsl*, and *TnfRSF12* are all synonyms. Such relationships could be recognized in a preprocessing step based on a database of synonyms that might be compiled using methods such as are described in (Yu and Agichtein, 2003; Yu et al., 2002). Second, the algorithm is generally unable to deal effectively with abbreviations, as in *GT* for *glucosyl transferase*, *glutamyl transpeptidase*, or *glutathione transferase*. Again a database of full forms and their abbreviations could be compiled, but then there is the additional problem of the ambiguity of abbreviations, and even the observation that what may in one case be an abbreviation of a gene name may in another context be an abbreviation for something unrelated to genes or proteins, e.g., *CAT* abbreviates *chloramphenicol acetyl transferase*, but also *computed axial tomography*. While a clear cut answer cannot be based on the strings alone, yet a probabilistic approach seems possible. Finally, our algorithm does not distinguish the actual words that appear in a gene name and their potential relative importance beyond the frequency of the trigrams that compose the terms. However, there is semantic information in names that goes well beyond this rudimentary level. For example, the names *angiotensin* and *angiotensin receptor* represent quite distinct entities as signaled by the word *receptor*, even though the strings have a strong similarity. Several investigators (Franzen et al., 2002; Hanisch et al., 2003; Narayanaswamy et al., 2003) have developed descriptive categories of different types of words that are used in the names of genes/proteins. While such categories are used for named entity recognition in text they may also prove useful in a system which gives a more sensitive rating of the relatedness of two names. These are possible directions for future research.

Our immediate goal in this research was to develop a method for comparing and rating two gene names for relatedness. We plan to use this to provide an access portal to GenBank in which one may enter a putative gene name and retrieve the names (with accompanying GenBank, i.d.) that appear to be the closest. From these the user may select and access those GenBank records judged by him to be useful matches to his query. We currently have a prototype running *3!*-GRM/HMM-3M. A second goal of our future research is to employ the algorithm as a part of a named entity recognition system. Here a large accepted list of gene names would be used, and potential names extracted from text would be compared to the list. The level of match achieved would be used as evidence in judging a putative name.

Appendix A. List of query gene names

11beta hsd
17beta hsd
18s rna
1bb
a2m
abl
acetyl coenzyme
acf

ache
aconitase
acp1
acrosin
acyl coenzyme
adenosylmethionine decarboxylase
adrenergic receptor
adrenomedullin
ae1
ags
beta spectrin
bombesin receptor
bsk
c33
caga
cannabinoid receptor
casein kinase
ccr4
connexin43
coproporphyrinogen oxidase
d1 receptor
delta4
diacylglycerol kinase
dinitrogenase reductase
ef1
elastin
epoxide hydrolase
factor va
filgrastim
gamma endorphin
grp94
heparin cofactor
hiv receptor
hsf1
igf1
interferon receptor
keratin
kininogens
leu3
ltd4 receptor
lymphotoxin
map1b
mcm2
monoamine oxidase
mre11
mucins
ngfi
ntr
opsin
ornithine decarboxylase
oxytocin receptor
p50
pbx1
pgi2
pkd
protein p53
rar gamma
retinoblastoma protein
rmp
sev
somatomedins
spectrin
stat5a
succinate dehydrogenase
synapsin i
tcrb
thymidine kinase
tlr4
tnf beta
triosephosphate isomerase
troponin i
tyrosine aminotransferase
ucp
v alpha
vegf c
vh1
wnt
xanthine oxidase
xpa
ypt1
yy1
zeta

References

Altschul, S., Gish, W., Miller, W., Myers, E., Lipman, D., 1990. Basic local alignment search tool. J. Mol. Biol. 215, 403–410.

Ashburner, M., Lewis, S., 2002. On ontologies for biologists: the Gene Ontology—untangling the web. Novartis Found Symp. 247, 66–80, discussion 80–63, 84–90, 244–252.

Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Wheeler, D.L., 2003. GenBank. Nucleic Acids Res. 31 (1), 23–27.

Charniak, E., 1993. Statistical Language Learning. The MIT Press, Cambridge, Massachusetts.

Cohen, K.B., Acquaah-Mensah, G.K., Dolbey, A.E., Hunter, L., 2002. Contrast and variability in gene names. In: Proceedings of the Paper Presentation at Natural Language Processing in the Biomedical Domain, University of Pennsylvania, 11 July 2002.

Collier, N., Nobata, C., Tsujii, J., 2000. Extracting the names of genes and gene products with a hidden markov model. In: Proceedings of the Paper Presentation of the 18th International Conference on Computational Linguistics, COLING'2000.

Consortium, T.G.O., 2000. Gene ontology: tool for the unification of biology. Nature Genet. 25, 25–29.

Damashek, M., 1995. Gauging similarity with n-grams: Language-independent categorization of text. Science 267, 843–848.

de Bruijn, B., Martin, J., 2002. Getting to the (c)ore of knowledge: mining biomedical literature. Int. J. Med. Inf. 67 (1-3), 7–18.

de Bruijn, L.M., Hasman, A., Arends, J.W., 2000. Supporting the classification of pathology reports: comparing two information retrieval methods. Comput. Meth. Progr. Biomed. 62, 109–113.

Durbin, R., Eddy, S., Krogh, A., Mitchison, G., 1998. Biological Sequence Analysis. Cambridge University Press, Cambridge.

Frantzi, K., Ananiadou, S., Mima, H., 2000. Automatic recognition of multi-word terms: the C-value/NC-value method. Int. J. Digital Libraries 3, 115–130.

Franzen, K., Eriksson, G., Olson, F., Asker, L., Liden, P., Coster, J., 2002. Protein names and how to find them. Med. Informat. 67, 49–61.

Fukuda, K., Tsunoda, T., Tamura, A., Takagi, T., 1998. Toward information extraction: identifying protein names from biological papers. In: Proceedings of the Paper Presentation at the Pacific Symposium on Biocomputing, PSB'98.

Gotoh, O., 1982. An improved algorithm for matching biological sequences. J. Mol. Biol. 162 (3), 705–708.

Hahn, U., Honeck, M., Piotrowski, M., Schulz, S., 2001. Subword segmentation–leveling out morphological variations for medical document retrieval. Proc. AMIA Symp., 229–233.

Hanisch, D., Fluck, J., Mevissen, N.-T., Zimmer, R., 2003. Playing biology's name game: identifying protein names in scientific text. In: Proceedings of the Paper Presentation at the Pacific Symposium on Biocomputing, PSB'03.

Jacquemin, C., 1994. FASTR: A unification-based front-end to automatic indexing. In: Proceedings of the Paper Presentation at the RIAO 94, Rockefeller University, New York, NY.

Jacquemin, C., 2001. Spotting and Discovering Terms through Natural Language Processing. The MIT Press, Cambridge, MA.

Kim, W.G., Wilbur, W.J., 2001. Corpus-based statistical screening for content-bearing terms. J. Am. Soc. Informat. Sci. 52 (3), 247–259.

Krauthammer, M., Rzhetsky, A., Morozov, P., Friedman, C., 2000. Using BLAST for identifying gene and protein names in journal articles. Gene 259, 245–252.

Liu, H., Friedman, C., 2003. Mining terminological knowledge in large biomedical corpora. Pac. Symp. Biocomput., 415–426.

Manning, C.D., Schutze, H., 1999. Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, MA.

McCray, A.T., Srinivasan, S., Browne, A. C., 1994. Lexical methods for managing variation in biomedical terminologies. In: Proceedings of the Paper Presentation at the 18th Annual Symposium on Computer Applications in Medical Care.

Mitchell, T.M., 1997. Machine Learning. WCB/McGraw-Hill, Boston.

Narayanaswamy, M., Ravikumar, K.E., Vijay-Shanker, K., 2003. A biological named entity recognizer. In: Proceedings of the Paper Presentation at the Pacific Symposium on Biocomputing, PSB'03.

Needleman, S.B., Wunsch, C.D., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48 (3), 443–453.

Nenadic, G., Mima, H., Spasic, I., Ananiadou, S., Tsujii, J.-I., 2002. Terminology-driven literature mining and knowledge acquisition in biomedicine. Med. Informat. 67, 33–48.

Nenadic, G., Spasic, I., Ananiadou, S., 2003. Terminology-driven mining of biomedical literature. Bioinformatics 19 (8), 938–943.

Proux, D., Rechenmann, F., Julliard, L., Pillet, V., Jacq, B., 1998. Detecting gene symbols and names in biological texts: a first step toward pertinent information extraction. In: Proceedings of the Paper Presentation at the Ninth Workshop on Genome Informatics.

Pruitt, K., Maglott, D., 2001. RefSeq and LocusLink: NCBI gene-centered resources. Nucl. Acids Res. 29, 137–140.

Rabiner, L.R., 1989. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE 77 (2), 257–286.

Salton, G., 1989. Automatic Text Processing. Addison-Wesley Publishing Company. Reading, MA.

Salton, G., 1992. The state of retrieval system evaluation, information processing. Informat. Proc. Manag. 28 (4), 441–449.

Salton, G., 1995. Letter. Science 268 (June 9), 1418–1419.

Smith, L., Yeganova, L., Wilbur, W.J., 2003. Hidden Markov models and optimized sequence alignments. Computat. Biol. Chem. 27, 77–84.

Tanabe, L., Wilbur, W.J., 2002. Tagging gene and protein names in biomedical text. Bioinformatics 18, 1124–1132.

Tanabe, L., Wilbur, W.J., 2004. Generation of a large gene/protein lexicon by morphological pattern analysis. J. Bioinformat. Computat. Biol. 2 (1), 1–16, in press.

TREC-Program-Committee, 1995. Lett. Sci. 268 (June 9), 1417–1418.

Wilbur, W.J., Kim, W., 2001. Flexible phrase based query handling algorithms. In: Proceedings of the Paper Presentation at the ASIST 2001 Annual Meeting, Washington, DC.

Yu, H., Agichtein, E., 2003. Extracting synonymous gene and protein terms from biological literature. Bioinformatics 19 (Suppl 1), I340–I349.

Yu, H., Hatzivassiloglou V., Friedman C., Rzhetsky A., Wilbur W.J., 2002. Automatic extraction of gene and protein synonyms from MEDLINE and journal articles. Proc. AMIA Symp., 919–923.

Yu, H., Hatzivassiloglou, V., Rzhetsky, A., Wilbur, W.J., 2002. Automatically identifying gene/protein terms in MEDLINE abstracts. J. Biomed. Inform. 35 (56), 322–330.